The Formal Language and Design Principles of Autonomous DNA Walker Circuits

Michael A. Boemo,[†] Alexandra E. Lucas,[†] Andrew J. Turberfield,[*,†] and Luca Cardelli[*,‡,¶]

[†]*University of Oxford Department of Physics*

[‡]*University of Oxford Department of Computer Science*

[¶]*Microsoft Research Cambridge*

E-mail: Andrew.Turberfield@physics.ox.ac.uk; luca@microsoft.com

Abstract

Simple computation can be performed using the interactions between single-stranded molecules of DNA. These interactions are typically toehold-mediated strand displacement reactions in a well-mixed solution. We demonstrate that a DNA circuit with tethered reactants is a distributed system and show how it can be described as a stochastic Petri net. The system can be verified by mapping the Petri net onto a continuous-time Markov chain, which can also be used to find an optimal design for the circuit. This theoretical machinery can be applied to create software that automatically designs a DNA circuit, linking an abstract propositional formula to a physical DNA computation system that is capable of evaluating it. We conclude by introducing example mechanisms that can implement such circuits experimentally and discuss their individual strengths and weaknesses.

Computation with DNA has been the subject of much interest from the points of view of both computer science and nanomedicine. A 2009 paper by Andrew Phillips and Luca

Cardelli showed how DNA strand displacement can be thought of as a formal computing language.[1] Further work by Matthew Lakin and colleagues produced Microsoft Visual DSD, a computational tool for the design and analysis of such reactions.[2] In the field of nanomedicine, Benenson et al. created a biomolecular DNA computing system that can produce an mRNA inhibitor to control gene expression.[3]

These papers all consider DNA strands as freely floating reactants in a well-mixed solution. There are no topological or geometric constraints that prevent two species from interacting. Such constraints can be introduced by tethering DNA reactants to rigid structures. Peng Yin and colleagues designed a DNA Turing machine that operates by DNA walkers moving on a rigid lattice.[4] In a 2005 paper, Jonathan Bath and colleagues introduced a "burnt-bridges" DNA walker powered by a nicking enzyme that is capable of traversing a track of single-stranded DNA anchorages (Figure 1).[5] Shelley Wickham and colleagues built on this design in a 2011 paper[6] that demonstrated direct observation of a DNA walker's stepwise movement on a DNA origami tile.[7] They also showed how the walker could be programmed to navigate a series of tracks.[8] The result was a DNA walker that could perform a computation, namely a binary decision tree. This is a "local" computation that is performed by reactants that are tethered to the origami tile.

Localized DNA computation has also been the subject of theoretical and computational study. A recent paper by Dannenberg et al. analyzed the computational potential of localized DNA circuits.[9] Microsoft Visual DSD can aid the experimental design of DNA strand displacement systems by performing stochastic simulation and detecting reaction leaks.[2] Lakin and colleagues recently incorporated tethered DNA reactants into Microsoft Visual DSD, allowing for topological and geometric constraints in DNA circuits.[10]

We demonstrate how localized DNA circuits can be analyzed as distributed systems. As such, they can be automatically designed and verified by software. The input to the localized DNA circuit can be posed using the language and grammar of the propositional calculus (Section 1). This input is then abstracted into a directed graph that captures

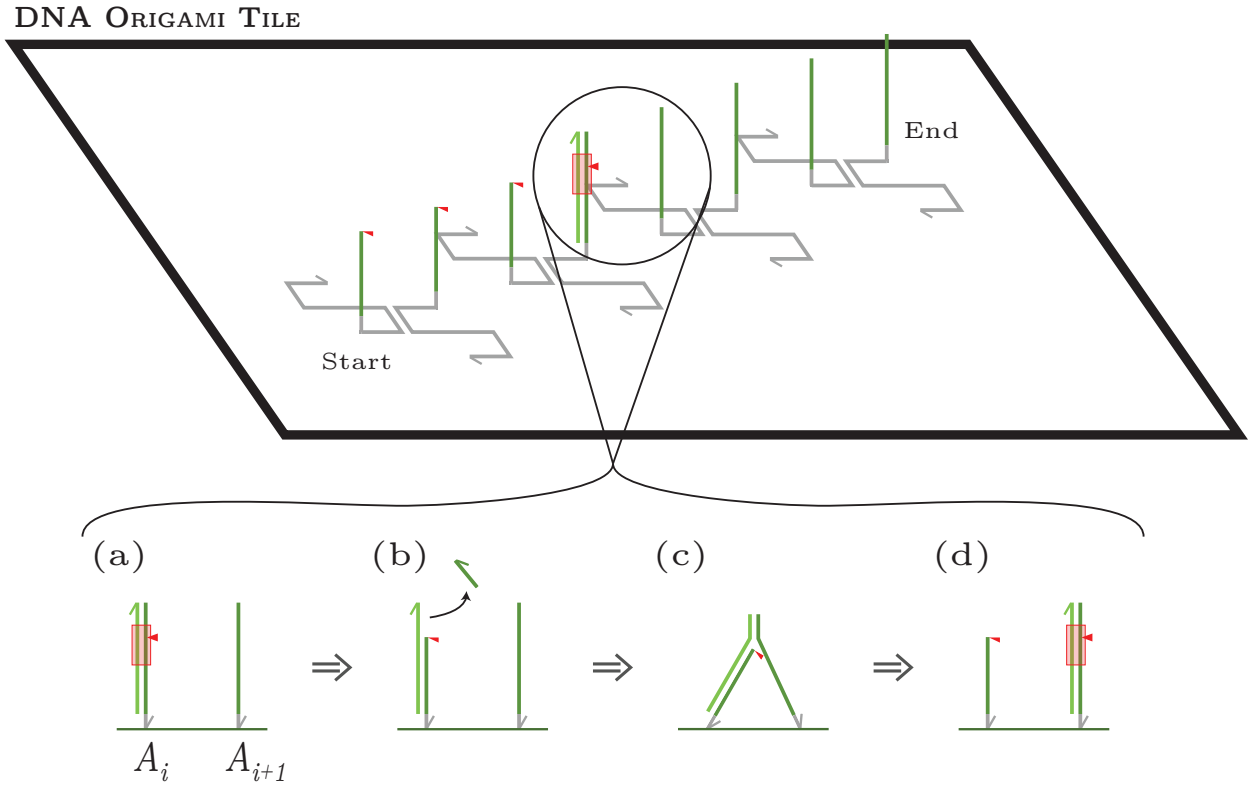

Figure 1: The "burnt-bridges" DNA walker mechanism.[5] Single-stranded oligonucleotides are shown as half arrows, with the arrowhead indicating the 3' end. (a) A single-stranded DNA walker is bound to a complementary single-stranded anchorage (A_i), completing the restriction site (red box) for a nicking enzyme. (b) The nicking enzyme cuts the 5' end of anchorage A_i at the location shown by the red triangle, exposing a short toehold on the 3' end of the DNA walker. (c) The toehold on the walker binds to anchorage A_{i+1}. (d) The DNA walker moves from anchorage A_i to A_{i+1} via a toehold-mediated branch migration reaction. By continuously repeating these steps, a DNA walker can navigate down a track of single-stranded anchorages. The cut toeholds inhibit the walker from stepping backwards, creating unidirectional motion.

the topology of the circuit (Section 2). Because the localized DNA circuit is a distributed system, it can be modeled as a stochastic Petri net. Analysis of this Petri net determines the geometry of the circuit (Section 3). Examples of DNA mechanisms that can implement these Petri nets are described and their strengths and weaknesses discussed (Section 4).

Results and Discussion

The Propositional Calculus of DNA Localized Computation Circuits

If P is a set of propositional variables (also known as primitive symbols or atomic formulae) and Ω is the set of all logical connectives, then there is an alphabet for the propositional calculus defined by $\Sigma = P \cup \Omega$. If Σ^* (where $*$ is the Kleene star) is the set of all words over Σ, then the language of the propositional calculus is a subset $L \subset \Sigma^*$ of words that are well-formed according to rules of the grammar.

Localized DNA computation systems can be designed to evaluate propositional formulae, and their action can be written in the formal language of the propositional calculus. The set of all logical connectives Ω can be partitioned into disjoint subsets according to their arity, or the number of arguments each connective takes:

$$\Omega = \Omega_0 \cup \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_n.$$

For localized DNA computation systems, attention is restricted to nullary, unary, and binary logical connectives where,

$$\Omega_0 = \{\mathbf{0}, \mathbf{1}\},$$
$$\Omega_1 = \{\neg\},$$
$$\Omega_2 = \{\vee, \wedge, \rightarrow, \leftrightarrow\}.$$

The rules of the propositional calculus can be used to search for a simpler form of a propositional formula in order to make the corresponding DNA computation system more efficient. An example of a word in Σ^* is the propositional formula

$$(x \wedge y) \vee (x \wedge z). \tag{1}$$

It has three logical connectives, and hence will require three logic gates. In this case, it is possible to use the distributive rule from the propositional calculus to find an equivalent form that requires only two:

$$(x \wedge y) \vee (x \wedge z) \equiv x \wedge (y \vee z). \tag{2}$$

There are libraries available that can implement heuristics for such a search, e.g. SymPy.[11]

Directed Graph Abstraction

An input propositional formula, such as those described in the previous section, can be used to find the topology for a localized DNA circuit. Every propositional variable is represented by a track, or linear array of DNA anchorages tethered to an origami tile, and a DNA walker. If the propositional variable holds the value **1**, then the walker will begin walking at the start of its track. Tracks that always take the value **1** have a DNA walker that will always start walking.

DNA walkers are able to perform universal Boolean logic if they are able to block other walkers on tracks that intersect their own. It is straightforward to construct a NOR gate, which is a functionally complete operator, using track blockage (Figure 2, upper left). Figure 2 shows both an AND gate constructed out of NOR gates and an alternative design that is simpler and uses fewer tracks.

A formula written in the propositional calculus, together with the chosen design for each gate, is enough to completely determine the topology of a corresponding DNA walker circuit.

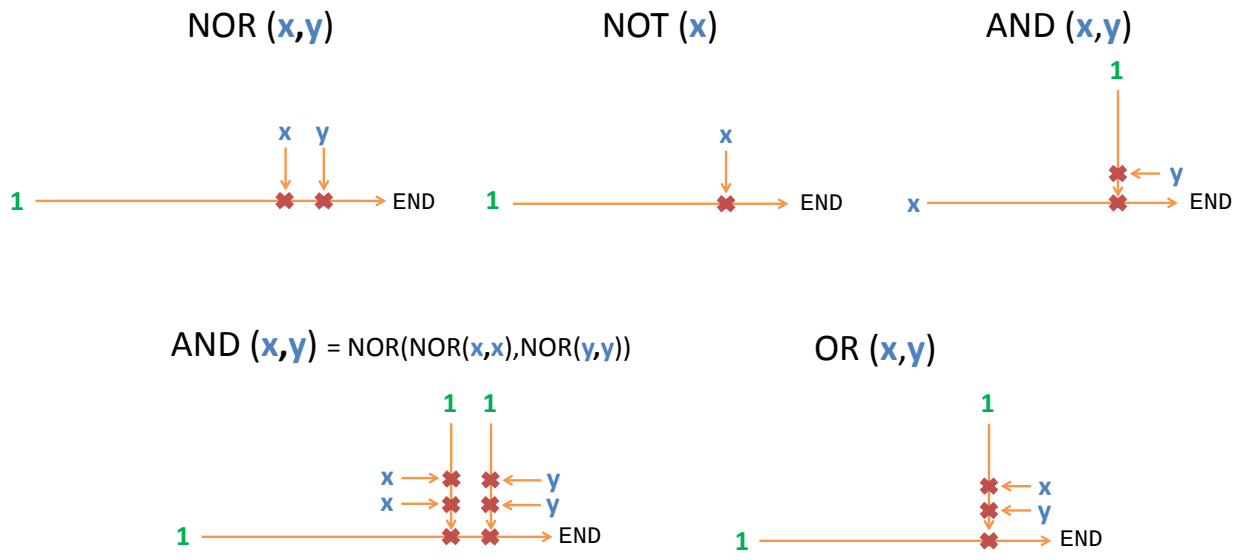

Figure 2: Track diagrams showing possible choices of design for NOR, NOT, OR, and AND logic gates that use the interaction (blockage) between DNA walkers. Each walker stays on its own track, and all walkers begin stepping at the same time. Walkers denoted by **1** will always walk while **x** and **y** are walkers that will only start walking if their respective propositions are true. Walkers can block another track at the junctions marked with red crosses. The gate evaluates to **1** if the walker whose track has END at the end of it is indeed able to make it to the end without being blocked.

In the context of DNA localized computation systems, topology refers to the "connectedness" of the tracks. Two tracks are said to be topologically connected if they intersect such that the walkers on both tracks can interact with each other.

More formally, for any chosen gate design, it is possible to describe the topology of each gate in terms of a directed graph $G_D = (V, E)$ where V is a set of vertices and E is a set of edges represented as ordered pairs. For the NOR gate,

$$V = \{\mathbf{1}, x, y\}\,,\ E = \{(x, \mathbf{1}), (y, \mathbf{1})\}\,.$$

The set V can be interpreted as the gate having three tracks, one for each of walkers $\{\mathbf{1}, x, y\}$. The set of ordered pairs E indicates that the x walker blocks the $\mathbf{1}$ walker and the y walker blocks the $\mathbf{1}$ walker. Figure 2 shows one possible choice of gate design. Once a choice is made, directed graphs can be constructed for each gate as shown in Table 1. Directed graphs for the gates can be pieced together to form one directed graph for the whole circuit, capturing the topology of a DNA circuit that evaluates the propositional formula.

Table 1: The sets of vertices V and edges E in the directed graph that correspond to each logic gate. Subscripts are used to differentiate between unique tracks.

	V	E
NOR	$\{\mathbf{1}, x, y\}$	$\{(x, \mathbf{1}), (y, \mathbf{1})\}$
NOT	$\{\mathbf{1}, x\}$	$\{(x, \mathbf{1})\}$
OR	$\{\mathbf{1}_1, \mathbf{1}_2, x, y\}$	$\{(\mathbf{1}_2, \mathbf{1}_1), (x, \mathbf{1}_2), (y, \mathbf{1}_2)\}$
AND	$\{\mathbf{1}, x, y\}$	$\{(\mathbf{1}, x), (y, \mathbf{1})\}$

Adding the directed graph abstraction between the propositional formula and its resulting track diagram has a number of advantages. At the topological level, the system becomes easier to analyze and simplify by automated means. The most immediate example is detecting certain redundancies, which can informally be thought of as double negatives. For any tracks a and b, if there exists a path $\{(a, \mathbf{1}_i), (\mathbf{1}_i, \mathbf{1}_j), (\mathbf{1}_j, b)\} \subset E$, then we may replace this path by $\{(a, b)\}$. In logical terms, this is the equivalent of writing $\neg(\neg b) = b$.

A key use of the directed graph structure is that it can represent circuits that cannot be posed in the propositional calculus. The most immediate example is FANOUT, which can be written as a directed graph:

$$V = \{x, \mathbf{1}_1, \mathbf{1}_2, \mathbf{1}_3, \mathbf{1}_4\},$$
$$E = \{(x, \mathbf{1}_1), (x, \mathbf{1}_2), (\mathbf{1}_1, \mathbf{1}_3), (\mathbf{1}_2, \mathbf{1}_4)\}.$$

As shown in Figure 3, FANOUT requires an additional property of the walker-track system: the walker must be able to block a track and keep walking, blocking additional tracks thereafter. This property, as well as the FANOUT gate itself, is not necessary to perform universal Boolean logic. It can, however, be useful in simplifying track designs by using fewer walkers overall.

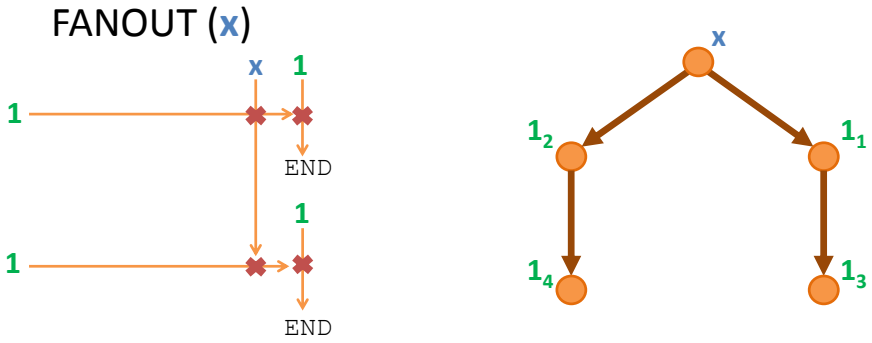

Figure 3: A track diagram of a possible design choice for a FANOUT gate (left) along with its directed graph abstraction (right). This design requires a walker that can block a track and keep walking, blocking other tracks thereafter.

Localized DNA Circuits as Distributed Systems

This section shows how introducing another piece of information, a tolerance for the probability of error, allows one to use the circuit topology to find a circuit geometry. A geometry defines the length of each track and specifies the locations where the tracks intersect. The tools needed to find this information come from analyzing the localized DNA circuit as a

distributed system.

A distributed system is a network of autonomous computers that can perform a coordinated action by passing messages between different computers in the network. When the anchorages on an origami tile are viewed as networked computers and the walker is viewed as the the message, a localized DNA computation system becomes a distributed system. The advantage of looking at the DNA circuit in this light is that it can be readily represented and analyzed as a Petri net.

The "burnt-bridges" walker-track system can be modeled as shown in Figure 4, upper. The initial marking shows the DNA walker, represented by a token, on the first anchorage G1. The stochastic Petri net model gives each walker a rate at which it steps forward onto the next anchorage. The transition from the first anchorage G1 to the second anchorage G2 fires at the rate at which the walker steps from one anchorage to the next. Such transitions require two tokens to fire. The tokens in the bottom row of nodes are used up as the walker steps forward, so another walker will not be able to step down the track after the current walker has finished. Physically, this bottom row of nodes represents the 5' end of the anchorages that are irreversibly cut by the nicking enzyme. A reusable track can be represented in a similar fashion (Figure 4, lower). In both cases, walkers are assumed to only step forward and to remain on the last anchorage once they reach the end of their track.

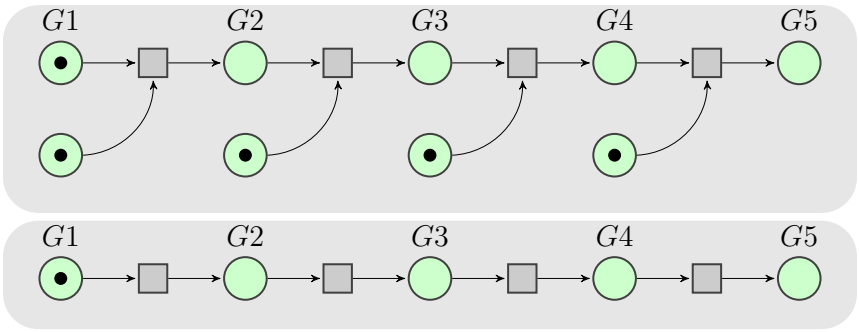

Figure 4: *upper* Stochastic Petri net representation of the "burnt-bridges" walker.[5] This track can only be used once, as the tokens in the bottom row of nodes are used up as the walker steps. *lower* Stochastic Petri net of a reusable track. Each transition only requires one token to fire, and no tokens are used up in the process.

Junctions between tracks are needed to implement the entire localized DNA circuit as a stochastic Petri net. Figure 5 shows a Petri net where a designated blocking walker (blue walker) can block another walker (green walker) if the blue walker arrives at the junction first. If the green walker arrives at the junction first, it steps through to the end of its track as normal.

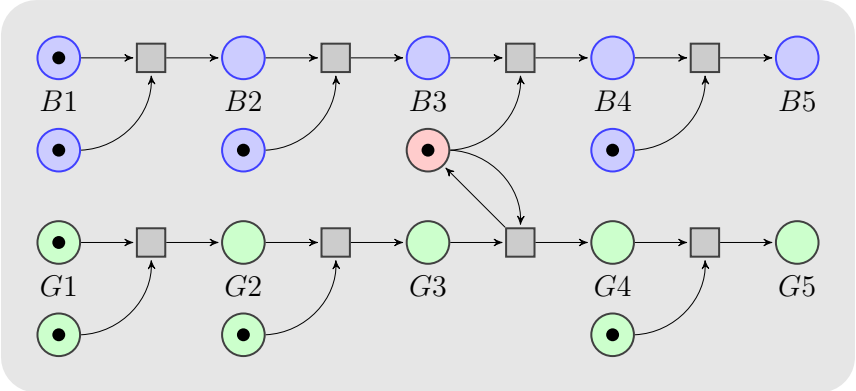

Figure 5: Two tracks, green and blue, with a blocking junction on the third anchorage of each track (G3 and B3). If the blue walker arrives at the junction first, it can block the green track by using up the token of the shared node (shown in red). Blocking is not symmetric: the blue walker can block the track for the green walker, but not vice versa.

A stochastic Petri net can be mapped directly onto a Markov process. If the DNA system can be posed as a Petri net, then it can be mapped onto a continuous-time Markov chain (CTMC). Using this CTMC, the probability of certain properties of the system can be computed using techniques from formal verification.

The localized computation systems discussed thus far operate under the assumption that all DNA walkers, if they walk at all, begin walking at the same time and walk at the same rate. This imposes certain length restrictions on the tracks. In the NOT gate, for example, the track for the **1** walker must be sufficiently longer than the track for the x walker so that the **1** walker does not arrive at the junction first. If it does, a missed-chance error has occurred because the x walker has missed its chance to block the **1** walker.

By representing the DNA system as a Markov chain and analyzing each possible combination of track lengths, one can search for a design that is optimal in the sense that it

is compact and minimizes the probability of missed-chance errors. Starting from the state corresponding to the initial marking of the Petri net, the system is allowed to evolve according to the CTMC. It is assumed that all walkers can only step forward, if they step at all, and that walkers do not move forward once they are blocked or reach the end of their track. Hence, the Markov chain is an absorbing Markov chain, and an absorbing state will always be reached if the system runs to equilibrium. The absorbing states can be divided into two groups: one group that corresponds to a missed-chance error for at least one junction, and another group that corresponds to correct operation. Analysis of the CTMC determines the probability with which the system ends up in a missed-chance error state or a correct state after it is allowed to run for a long time.

Prism provides a natural scripting language for representing complex systems as continuous-time Markov chains.[12] It also allows the user to evaluate the probability of certain conditions, such as the probability of eventually ending up in a certain state. For example, if a walker B is intended to block a walker G, Prism can check the following property:

```
P = ?  [B=end & G>intersection].
```

In Prism's language, this is the probability that the G walker has moved through the junction before the B walker has arrived to block it. This statement measures the probability of a missed-chance error for that junction. Due to the modular nature of the Prism language, the code can be automatically generated from the directed graphs described in the previous section.

Model checking can be used to determine the system with the shortest tracks that still has a missed-chance error below a given tolerance. Finding this balance is critical: a compact system is easier to design and fit onto an origami tile, but tracks that are too short will cause missed-chance errors. The output is the assignment of a natural number to each track corresponding to the smallest number of anchorages needed to stay within the specified

tolerance for the missed-chance error. The assumption that a track is always blocked on its penultimate anchorage, or as close as possible if the track is blocked twice, is sufficient to determine the track geometry of the system.

Example

This theoretical machinery forms the foundation for software that can design track systems. The only inputs required are a propositional formula, a choice of gate design, and a tolerance for the missed-chance error. The plot in Figure 6 was automatically generated in this way.

The simplified propositional formula in Equation 2 can be arranged into a parsing tree based on its logical connectives. Prism can be used to find the lengths (in anchorages) of each track that minimizes the missed-chance error. Using a tolerance of 0.15 probability for the missed-chance error results in the following optimized track lengths:

$$\mathbf{1}_1 = 14 \text{ anchorages}, \mathbf{1}_2 = \mathbf{1}_3 = 7 \text{ anchorages}, x = y = z = 2 \text{ anchorages}.$$

Using the lengths of each individual track shown above and the blocking topology from the directed graph, a track design is generated that evaluates the original propositional formula (Figure 6).

DNA Mechanisms for Track Blockage

There are a number of different strategies that can be employed to block a track, each with advantages and disadvantages. We discuss three of them here: destroying the green track, removing the green walker from the track system, and trapping the green walker at the junction. Example mechanisms for each strategy based on the “burnt-bridges" walker are described below. They all implement the Petri net in Figure 5.

The junction anchorage located where the two tracks intersect acts as a transducer, enabling the blue walker to convert the junction anchorage from an accept state to a blocking

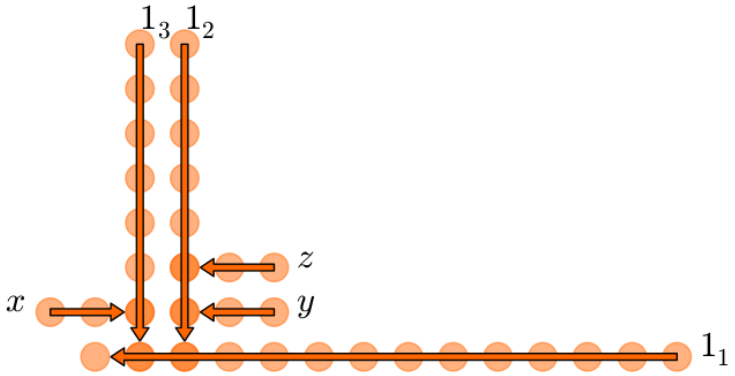

Figure 6: Final design of the DNA localized computation system that can evaluate Equation 2. Each anchorage is represented by a light orange circle. The individual tracks, along with their directionality, are indicated by orange arrows. Each track is labelled at its starting anchorage by the name of the walker that walks along it.

state. A blue walker can block the track for the green walker if the blue walker is present and arrives at the junction first. The notation in Figures 7-9 is identical to Figure 1. Walkers and tracks are identified by color (blue or green) and walkers are assumed to not step onto the wrong track. This can be guaranteed by giving the two walkers orthogonal base sequences. A completed restriction site is depicted by a red box and the nicking site for the enzyme is indicated by a small red triangle. Some mechanisms require the nicking enzyme to avoid cutting certain domains that closely resemble the restriction site: cutting is prevented in these cases by a restriction site mismatch, where one nucleotide in the restriction site is altered. Domains with a restriction site mismatch are indicated by a black cross.

Destroying the Track

A blocking mechanism based on destruction of the track is shown in Figure 7. There is a tall anchorage at the junction of the two tracks that has a green binding site at its "top" (farthest from the origami tile) and a blue binding site at its "bottom" (nearest the origami tile). When this anchorage is intact, then the junction acts as a normal green anchorage and the green walker steps onto the green binding site at the top of the junction anchorage.

If the blue walker arrives at the junction first, it is able to block the green walker. It

hybridizes to its binding site at the bottom of the junction. When the enzyme cuts the junction anchorage, the top of the junction anchorage (containing the green binding site) diffuses away from the tile. If the green walker subsequently arrives at the junction, the absence of a binding site on the junction anchorage prevents it from reaching the end of the track.

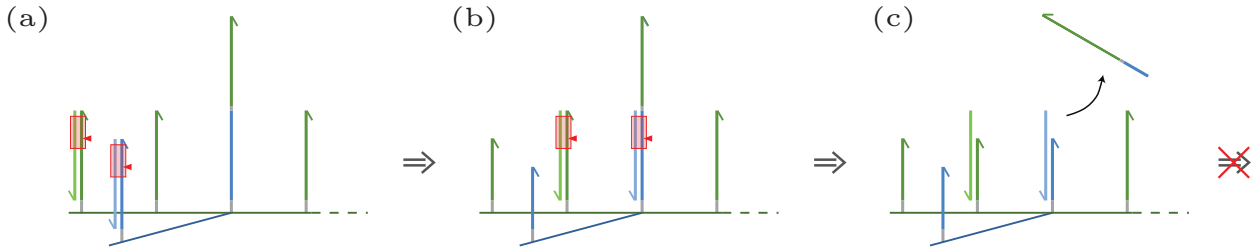

Figure 7: An example of a DNA mechanism that blocks by destroying the track. (a) The blue walker is likely to arrive at the junction first to block the track for the green walker. (b) By hybridizing to its binding site near the bottom of the junction anchorage, the blue walker completes a restriction site for the nicking enzyme on the junction. (c) The enzyme cuts the junction, causing the top portion of the junction (which contains the green binding site) to diffuse away from the track. Without the junction binding site, the green walker is unable to reach the end of its track and is blocked.

A key advantage is that the blue walker can continue stepping down its track after blocking the green track, potentially blocking subsequent tracks. This type of mechanism would be ideal for constructing a FANOUT gate. However, the long junction anchorage can introduce two possible errors into the system. First, the extended length of the junction anchorage means that the junction can bind to the green walker before it has reached the adjacent anchorage, increasing the chance of a missed-chance error. Second, the freely diffusing top portion of a cut junction anchorage can bind to a green walker, removing it from the track and potentially redepositing it at the end of the track. Both of these errors can be alleviated by shortening the length of the green toehold domain on the junction anchorage, lowering its binding affinity to the green walker.

Different origami structures have different degrees of flexibility. In some cases, the tile may bend, allowing the blocked green walker and the final anchorage of the green track to

come close together. This can cause a slow leak whereby a blocked walker can still reach the end of its track. When this system is implemented using a flexible tile, a gap can be introduced in the green track after the junction. If the green walker is unblocked, the long junction anchorage can help it cross this gap and reach the end of the track. When the green walker is blocked, the gap prevents the blocked walker from reaching the end of the track when the tile flexes.

Removing the Walker

In this mechanism, the blue walker destroys the junction anchorage for the green walker and prepares a strand that removes it from the track. Figure 8 shows the blocking mechanism. The blue walker differs from the green walker in two important ways. First, the blue walker is in reverse orientation to the green walker; its toehold domain is at the 5' end. Second, the blue walker has an extra "tail" domain at the 3' end that plays a role in blocking the green walker. The junction anchorage consists of three strands: a tether strand attached to the origami tile, a remover strand that is initially fully bound to the tether, and an auxiliary strand, also bound to the tether, that has a single-stranded 5' extension that acts as a green anchorage. The remover strand has the same base sequence as the green anchorage, with the exception of a single-base mismatch in the restriction enzyme recognition sequence.

If the green track is unblocked, the green walker steps onto the green binding site on the auxiliary strand of the junction anchorage. Once the restriction site for the nicking enzyme is completed, the auxiliary strand is cut and the green walker steps on as normal. If the blue walker arrives first, it binds to the blue toehold domain on the tether strand and displaces the auxiliary strand, removing the binding site for the green walker. The 3' tail domain on the blue walker is then in competition with the remover strand for a binding site on the tether strand, partially exposing a green toehold domain on the remover strand. When the green walker arrives at the junction, it binds to this toehold initiating a strand-exchange reaction in which the green walker hybridizes to the remover strand and displaces it from the junction

anchorage. A restriction site mismatch in the remover strand prevents enzymatic cleavage of the remover-green walker duplex. Hence, the green walker is blocked by removing it from the track.

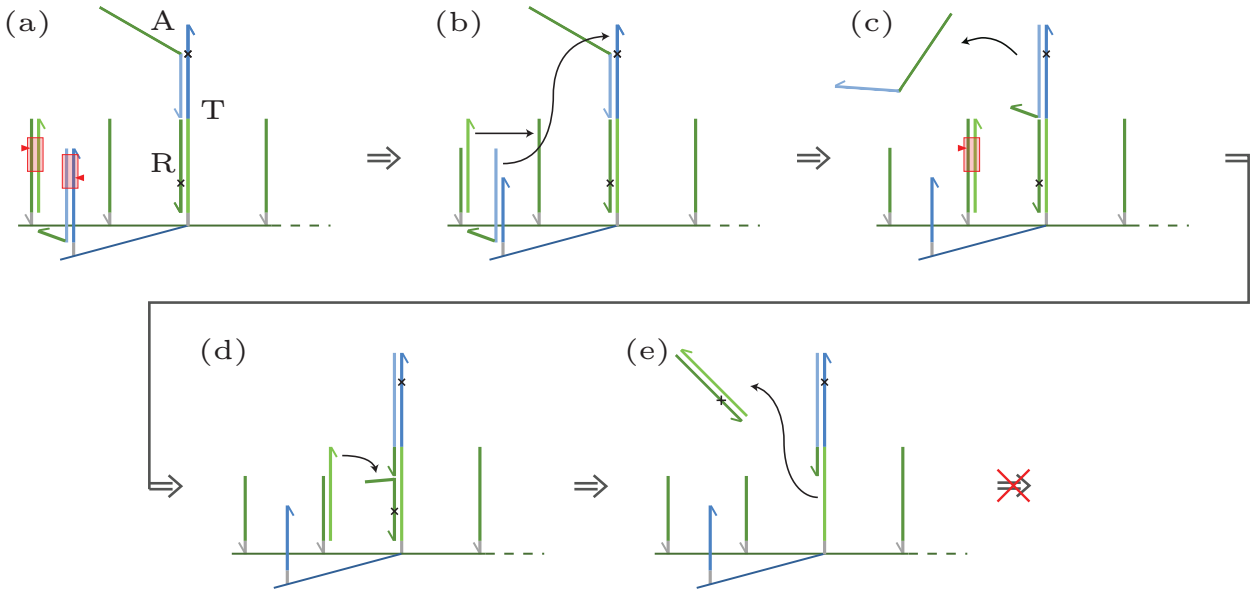

Figure 8: Removing the walker. The mechanism uses a three-strand junction: T = tether; A = auxiliary strand; R = remover. The blue walker displaces the auxiliary strand, exposing the toehold of the remover strand that can displace the green walker from the tile. (a) The blue walker is likely to arrive at the junction anchorage first. (b) The blue walker steps onto its binding site at the top of the tether strand, displacing the auxiliary strand. (c) The auxiliary strand diffuses away from the tile. (d) The tail domain on the blue walker exposes the green toehold of a remover strand. When the green walker arrives at the junction, it binds to the remover toehold that was exposed by the blue walker. (e) The green walker hybridizes to the remover strand, forming an inert duplex which is displaced from the track.

This blocking mechanism relies on the blue walker staying bound at the junction, so it cannot be used to make a FANOUT gate. The advantage of this mechanism is that the green walker is removed from the track, reducing the chance that flexibility of the tile will allow it to step over the junction. As with the track destroying mechanism described previously, shortening the green toehold domain in its binding site on the auxiliary strand at the junction can help prevent errors caused by extended reach. To scale this mechanism up to more complex circuits, the green walker should be able to both block a track and be blocked by another walker. For example, this is a necessary feature of the **1** walker in the

AND and OR gate (see Figure 2). This can be accomplished by giving the green walker a 5' tail domain so that it can act as a blocking walker at subsequent junctions.

Trapping the Walker

In this mechanism, shown in Figure 9, the blue walker enables the green walker to bind more stably at the junction than it can with the next anchorage in the track. Therefore, the green walker remains bound to the junction because it is the most energetically favorable state. The walkers have complementary single-stranded tail domains: the blue walker has a 3' tail domain while the green walker has a 5' tail domain.

If the green walker arrives first, it can hybridize to its binding site on the junction anchorage, the enzyme cuts the junction, and the green walker steps on as normal. However, if the blue walker arrives at the junction first, it steps onto its binding site at the top of the junction anchorage. When the green walker arrives, its 5' tail domain hybridizes to the complementary tail domain at the 3' end of the blue walker. Even when the enzyme cuts the junction anchorage, the green walker will not step onto the next anchorage because it is stabilized by the extra base pairing provided by the bound tail domain.

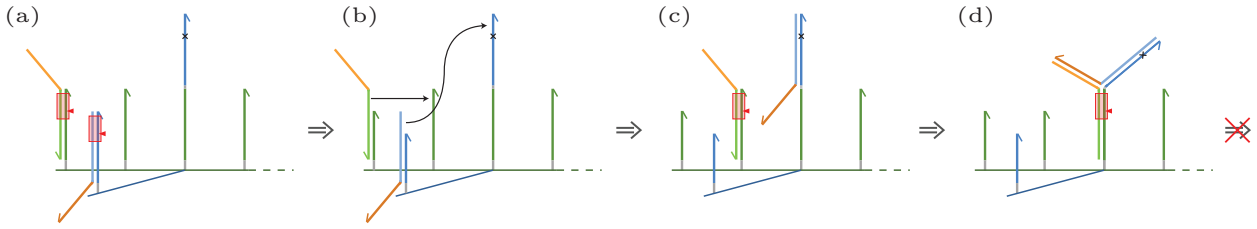

Figure 9: A mechanism that traps the green walker at the junction by stabilizing the walker-junction complex with extra base pairing. (a) The blue walker is likely to arrive at the junction first to block the green walker. (b) The blue walker steps onto to its binding site at the top of the junction anchorage. (c) The green walker arrives at the junction, ready to step onto its binding site at the bottom of the junction anchorage. (d) When the green walker hybridizes to the junction, the 5' tail on the green walker and the 3' tail on the blue walker hybridize to each other. The extra base pairing from the bound tail domains makes it more energetically favorable for the green walker to remain bound to the junction instead of stepping onto the next anchorage, even after the junction anchorage is cut.

The mechanism relies on the blue walker remaining at the junction to hybridize to the green walker, so this mechanism cannot be used to make a FANOUT gate. Unlike the previous two mechanisms, this mechanism is not hampered by the error caused by the extra reach of green binding site at the top of the long junction anchorage. For simplicity, Figure 9 shows the green walker with only one tail domain, the one used to trap it. If the circuit design requires the green walker to block subsequent tracks, it can be given a 3' tail (in addition to its 5' tail) that is the reverse complement of the tail domain on the walker it will block.

Conclusions

We have shown that localized DNA computation circuits can be analyzed as distributed systems. A propositional formula, a choice of gate design, and an error tolerance are enough to determine the geometry of the track system. There are several strategies that can be used to block a track, and each comes with its own advantages and disadvantages.

A challenge of working with DNA localized computation systems is making the system compact enough to fit on an origami tile while maintaining a low probability of missed-chance error at the junctions. We can imagine a logic gate on an origami tile that, if it evaluates to **1**, activates a messenger strand that can set another walker stepping on a neighboring origami tile. Such a mechanism would increase the scaling potential and reliability of localized DNA circuits.

The theory developed here can be improved upon by using it together with previously developed tools. One challenge in designing any DNA circuit is preventing unwanted interactions between components. As the circuit gets more complex, these leak reactions become more and more difficult to identify. Microsoft DSD already has automated means of detecting such leaks at the domain level. At the sequence level, NUPACK is an easy-to-use tool that can find suitable nucleotide sequences for a given design.[13] The three pieces of software can work together to first design a localized DNA circuit using the methods described above, then compile Microsoft DSD code to check for errors at the domain level, and finally compile

NUPACK code to generate nucleotide sequences for each strand.

Acknowledgement

The authors thank Andrew Phillips (Microsoft Research Cambridge), Marta Kwiatkowska (Oxford Computer Science), and Jonathan Bath (Oxford Physics) for guidance and helpful conversations. This research was supported by Engineering and Physical Sciences Research Council grants EP/G037930/1, EP/J500495/1 (AEL), Biotechnology and Biological Sciences Research Council grant BB/J00054X/1, the Marie Curie Initial Training Network EScoDNA (FP7 project 317110, MAB) and a Royal Society-Wolfson Research Merit Award (AJT).

Supporting Information Available

The software written to generate track designs is open source and freely available via the GitHub clone URL `https://github.com/MBoemo/DLCC.git`.

This material is available free of charge via the Internet at `http://pubs.acs.org/`.

References

(1) Phillips, A.; Cardelli, L. A programming language for composable DNA circuits. *J. R. Soc. Interface* **2009**, *6*, S419–S436.

(2) Lakin, M. R.; Youssef, S.; Polo, F.; Emmott, S.; Phillips, A. Visual DSD: A design and analysis tool for DNA strand displacement systems. *Bioinformatics* **2011**, *27*, 3211–3213.

(3) Benenson, Y.; Gil, B.; Ben-Dor, U.; Adar, R.; Shapiro, E. An autonomous molecular computer for logical control of gene expression. *Nature* **2004**, *429*, 423–429.

(4) Yin, P.; Turberfield, A. J.; Sahu, S.; Reif, J. H. In *DNA Computing*; Ferretti, C., Mauri, G., Zandron, C., Eds.; Lecture Notes in Computer Science; Springer Berlin Heidelberg, 2005; Vol. 3384; pp 426–444.

(5) Bath, J.; Green, S. J.; Turberfield, A. J. A free-running DNA motor powered by a nicking enzyme. *Angewandte Chemie* **2005**, *117*, 4432–4435.

(6) Wickham, S. F. J.; Endo, M.; Katsuda, Y.; Hidaka, K.; Bath, J.; Sugiyama, H.; Turberfield, A. J. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature Nanotechnology* **2011**, *6*, 166–169.

(7) Rothemund, P. W. K. Folding DNA to create nanoscale shapes and patterns. *Nature* **2006**, *440*, 297– 302.

(8) Wickham, S. F. J.; Bath, J.; Katsuda, Y.; Endo, M.; Hidaka, K.; Sugiyama, H.; Turberfield, A. J. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology* **2012**, *7*, 169–173.

(9) Dannenberg, F.; Kwiatkowska, M.; Thachuk, C.; Turberfield, A. J. In *DNA Computing and Molecular Programming*; Soloveichik, D., Yurke, B., Eds.; Lecture Notes in Computer Science; Springer International Publishing, 2013; Vol. 8141; pp 31–45.

(10) Lakin, M.; Petersen, R.; Gray, K. E.; Phillips, A. In *DNA Computing and Molecular Programming*; Murata, S., Kobayashi, S., Eds.; Lecture Notes in Computer Science; Springer International Publishing, 2014; Vol. 8727; pp 132–147.

(11) Joyner, D.; Čertík, O.; Meurer, A.; Granger, B. E. Open Source Computer Algebra Systems: SymPy. *ACM Commun. Comput. Algebra* **2011**, *45*, 225–234.

(12) Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of probabilistic real-time systems. *Proc. 23rd International Conference on Computer Aided Verification* **2011**, *6806*, 585–591.

(13) Zadeh, J. N.; Steenberg, C. D.; Bois, J. S.; Wolfe, B. R.; Pierce, M. B.; Khan, A. R.; Dirks, R. M.; Pierce, N. A. NUPACK: Analysis and design of nucleic acid systems. *J Comput Chem* **2011**, *32*, 170–173.